Empezando con Docker

Fernando Furtado

19/08/2022

```
~ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
b6f892c0043b: Pull complete
55010f332b04: Pull complete
2955fb827c94: Pull complete
3deef3fcbd30: Pull complete
cf9722e506aa: Pull complete
Digest: sha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1aa8
Status: Downloaded newer image for ubuntu:latest
~
```

Cuando hablamos dedesarrollo de software es común tener diversos ambientes, por ejemplo:

- Desarrollo
- Prueba
- Homologación
- Producción

Y otra cosa común en el mundo del desarrollo es tener divergencias entre estos ambientes.

Quién nunca escuchó la frase:

```
<!--En mi maquina funciona!-->
```

En este artículo vamos a abordar Docker como una alternativa para minimizar esta divergencia.

Pero, ¿Qué es Docker?

Docker es un sistema de virtualización no convencional. Pero, ¿qué quiere decir esto? en virtualización convencionales tenemos un software instalado en la máquina *HOST* que gerencia las máquinas virtuales ej.: VirtualBox, VMWare, Parallels y etc...

Para cada máquina virtual tenemos una instalación completa del sistema operacional (S.O) que queremos virtualizar, además de tener el propio hardware virtualizado.

Por ejemplo si yo necesito de una biblioteca común para todas las máquinas virtuales, tengo que instalar en cada una de ellas.

Docker utiliza un abordage diferente, él utiliza el concepto de container. ¿Qué sería container?

Comprendiendo el concepto de containers

Si pensamos en transportes de cargas, el container fue una revolución en esta área. Antes de ellos el tiempo de carga y descarga en un barco era gigantesco y el trabajo se hacía manualmente. Sin contar (debido la quiebra o deterioración), desvio y otros problemas.

Con la llegada de los containers fue posible transportar mercadorías de una forma más segura, de fácil manipulación, con poca o sin necesitar mano de obra en la carga o descarga. Y es justamente eso lo que Docker intenta hacer con nuestros softwares.

Beneficios al utilizar containers de Docker

Imagine nuestro software como una mercadoria a ser transportada como por ejemplo, del ambiente de Desarrollo para Producción

Para hacer eso necesitamos garantizar que nuestro ambiente de Producción tenga todos los requisitos previos instalados, de preferencia una versión del S.O(sistema operacional)

parecida con la del ambiente de Desarrollo entre otros cuidados que deben ser tomados(relacionados con permisos, servícios dependientes, etc...).

Con Docker tenemos un container con nuestro software, este container es llevado entero para el otro ambiente.

De esta manera, no necesitaríamos preocuparnos con requisitos previos instalados en otros ambientes, o sea, versión del S.O(sistema operacional) permisos y si quisiéramos también podríamos tener containers para los servicios dependientes. De esta forma minimizamos mucho la divergencia entre los ambientes.

¿Cómo se haría esto en Docker?

Esta idea de container es bastante antígua, al principio Docker utilizaba internamente un proyecto llamado LXC (Linux Container).

El proyecto LXC usa en segundo plano diversas funcionalidades presentes en el Kernel de Linux. A continuación, colocaré una lista de algunas de estas funcionalidades:

- **chroot**- Responsable por mapear los directorios del S.O. y crear el punto de montaje (/,/etc,/proc entre otros).
- **cgroup**- Responsable por controlar los recursos por procesos. Con ello podemos por ejemplo limitar el uso de memoria y/o procesador para un proceso específico.
- **kernel namespace**- Responsable de aislar el proceso, punto de montaje entre otras cosas. Aislandolo, conseguimos la sensación de estar utilizando una máquina diferente de la máquina host. Por que vemos solamente el punto de montaje específico y procesos específicos, incluso nuestros procesos empiezan con PID bajo.
- **kernel capabilities**- Entre otras cosas, logramos ejecutar algunos comandos de forma privilegiada.

Hagamos una practica

Ahora que tenemos una noción de lo que es y para que utilizamos Docker, vamos a hacer el download de la herramienta y vamos a empezar en este mundo de los containers.

Instalando Docker

Actualmente Docker está disponible en dos versiones: Docker Community Edition(CE) y Docker Enterprise Edition(EE).

En ambas versiones tenemos acceso a todas las API, la principal diferencia entre las dos versiones es el perfil deseado de aplicación. En EE tenemos un ambiente homologado por Docker con todas la infraestructura certificada, segura y pensada para el mundo enterprise. En la versión CE podemos llegar al mismo nível que EE pero de una forma manual.

En este link puedes encontrar las distribuiciones para downloads en cada sistema operacional disponible y los pasos para su instalación.

Hecha la instalación, ejecute este comando en el termianal `docker --version`. Si la instalación es exitosa deberá aparecer en la pantalla, algo igual a esto `Docker version 17.03.1-ce, build c6d412e.`

Imágenes

Docker trabaja con el concepto de imágenes, o sea, para colocar un container en funcionamento este necesita tener la imagen en host.

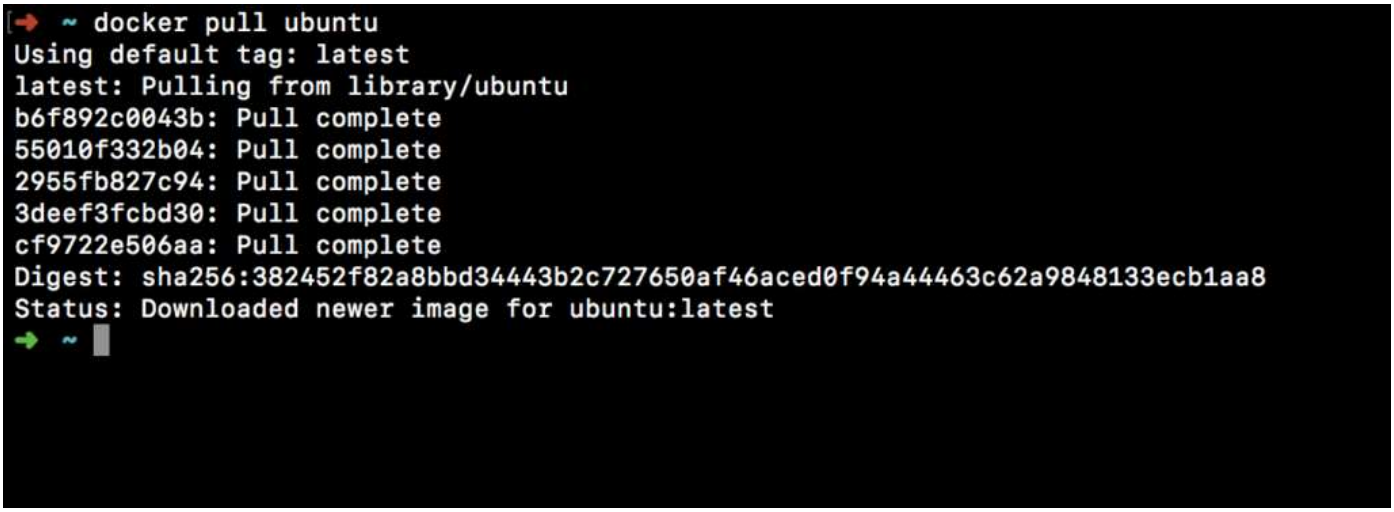
Las imágenes pueden ser bajadas de un repositorio (la nomenclatura para este repositorio es registry) o creadas localmente y compiladas. Este es el link para el registry del Docker.

En este registry podemos tener imágenes oficiales y no oficiales. Además podemos crear nuestras propias imágenes, también es posible hacer upload de ellas en un registry.

Bajando imágenes

Para bajar una imagen podemos usar el comando `docker pull` y el nombre de la imagen que queremos bajar. Vamos a bajar la imagen del *Ubuntu*, para eso ejecute el siguiente comando en el terminal: `docker pull ubuntu`.

```
[caption id="attachment_7091" align="aligncenter" width="750"]
```



Resultado

```
  docker pull ubuntu[/caption]
```

Aqui Docker bajó nuestra imagen. Perciban que una imagen es compuesta de varias capas, por este motivo tuve que hacer varios ***Downloads/Pulls***.

Listando imágenes bajadas

Para listar todas las imágenes podemos usar el comando `docker imagenes`. El retorno de ese comando es algo semejante a:

El nombre de la imagen es exhibido en la columna `REPOSITORY`, cada imagen tiene un identificador único que es exibido en la columna `` `IMAGEN ID ``. La columna `TAG` indica la "**versión**" de la imagen del `ubuntu`. El `latest`quiere decir que es la última "**versión**" de la imagen (la más reciente).

Ejecutando Containers

A partir de la imagen podemos iniciar cuantos containers queramos através del comando `docker run`.

Para accesar a un terminal de Ubuntu podemos utilizar el comando`docker run -i -t ubuntu`o `docker run -it ubuntu`. El parametro `-i` indica que queremos un container

interactivo, el `-t` indica que queremos anexar el terminal virtual `tty` del container a nuestro host.

Listando containers en ejecución

Para ver los containers en ejecución podemos utilizar el comando `docker ps`(en otro terminal o aba), y él exhibirá un retorno igual a este:

```
➜  ~ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
a6697ed945d5        ubuntu              "/bin/bash"         12 hours ago        Up 4 seconds                            dreamy_bassi
➜  ~
```

Aquí tenemos informaciones sobre los containers en ejecución, como id, imagen base, comando inicial, hace cuanto tiempo fue creado, status, cuáles puertas estan disponibles y/o mapeadas para el acceso y el nombre del mismo. Cuando no especificamos un nombre al iniciarlo, será generado un nombre aleatoriamente.

Cuando cerramos un container dejará de ser exhibido en la salida del comando `docker ps`, pero no significa que el container ya no existe. Para verificar los containers existentes que fueran cerrados podemos utilizar el comando `docker ps -a` y tendremos una salida igual a esta:

```
➜  ~ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
a6697ed945d5        ubuntu              "/bin/bash"         12 hours ago        Exited (0) 4 seconds ago                        dreamy_bassi
➜  ~
```

Como el propio `status` del container informa, el mismo ya salió de ejecución y en este caso salió con el status `0` (o sea salió normalmente).

Removiendo containers

Para remover el container podemos utilizar el comando `docker rm` e informar el id del container o su nombre. En nuestro caso podemos ejecutar el comando `docker rm 43aac92b4c99` o `docker rm dreamy_bassi` para remover el container por completo.

Caso tengamos la necesidad de remover todos los containers (en ejecución o cerrados) podemos usar el comando `docker rm $(docker ps -qa)`. La opción `-q` del comando `docker ps` tiene como salida solamente los ids de los containers, esta lista de ids es pasada para el `docker rm` y de esta manera serán removidos todos los containers.

Solo será posible remover un container caso el mismo no esté en ejecución, de lo contrario tenemos que encerrar el container para removerlo.

¿Cómo son creadas las imágenes?

En este momento podemos pensar que Docker es mágico (y lo es...). A partir de una imagen él puede rodar uno o más containers con poco esfuerzo, ¿ pero cómo son creadas las imágenes?

Una imagen puede ser creada a partir de un archivo de definición llamado de `Dockerfile`, en este archivo usamos algunas directivas para declarar lo que tendremos en nuestra imagen. Por ejemplo, se miramos la definición de imagen del Ubuntu podemos ver algo igual a esto:

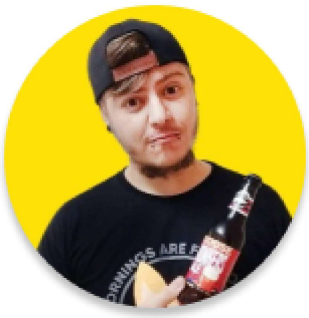
```
FROM scratch ADD ubuntu-xenial-core-cloudimg-amd64-root.tar.gz / . . . RUN mkd

CMD \["/bin/bash"\]
```

Para ver el Dockerfile completo consulte aquí.

Con un archivo `Dockerfile` podemos compilar el comando `docker build`. Al compilar el archivo `Dockerfile` creamos una imagen, pero eso es un tema para el proximo artículo.

Conozca nuestro curso de docker en Alura Latam.



Fernando Furtado

Fernando es desarrollador con enfasis en Java, mientras, ya hice proyectos para mobiles (iOS), donde actuó con lenguajes, como: Objective-C y Swift. En su trabajo, siempre consulta buenas practicas como pruebas o design de codigo (Design Pattern y SOLID, entre otros).

Este artículo fue adecuado para Alura Latam por: **Ingrid Cristina Pereira da Silva**

Cursos de DevOps

ARTÍCULOS DE TECNOLOGÍA > DEVOPS

En Alura encontrarás variados cursos sobre DevOps. ¡Comienza ahora!

SEMESTRAL

US$49,90

un solo pago de US$49,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 6 meses

CURSOS

Cursos de Programación
Lógica de Programación | Java

Cursos de Front End
HTML y CSS | JavaScript | React

Cursos de Data Science
Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps
Docker | Linux

Cursos de Innovación y Gestión
Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento